

Chapter 4

Sending Data to Your Application

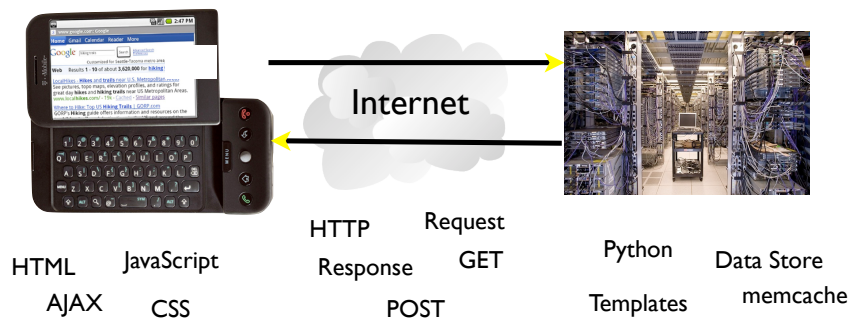
Charles Severance and Jim Eng
csev@umich.edu jimeng@umich.edu

Textbook: Using Google App Engine, Charles Severance



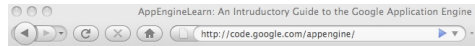
Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.
<http://creativecommons.org/licenses/by/3.0/>.

Copyright 2009, Charles Severance, Jim Eng



Making an HTTP request

- Connect to the server
 - a "hand shake"
- Request a document (or the default document)
 - GET <http://dr-chuck.com/page1.htm>
 - GET <http://www.mlive.com/ann-arbor/>
 - GET <http://www.facebook.com>



http://www.dr-chuck.com/page1.htm

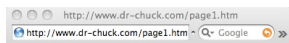
protocol host document



The First Page

If you like, you can switch to the [Second Page](#).

Go to "http://www.dr-chuck.com/page2.htm"



The First Page

If you like, you can switch to the [Second Page](#).

Go to "http://www.dr-chuck.com/page2.htm"

Browser

Web Server

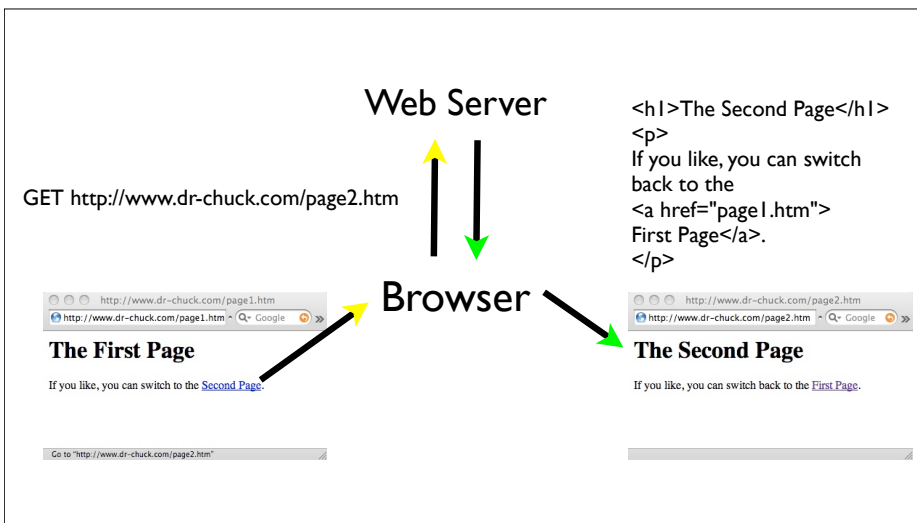
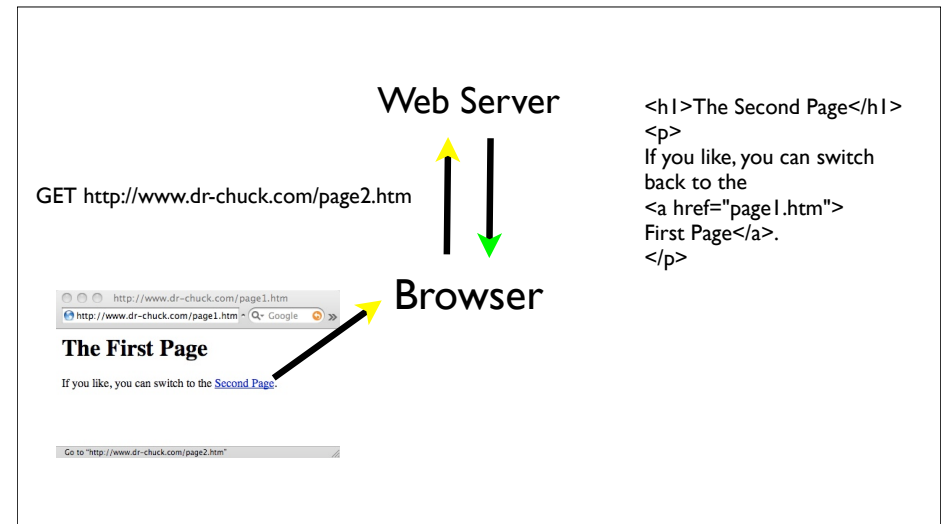
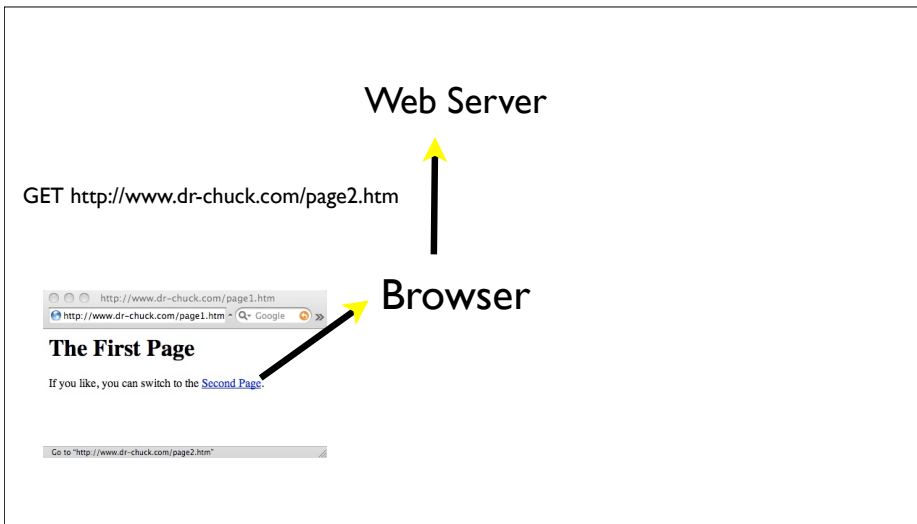


The First Page

If you like, you can switch to the [Second Page](#).

Go to "http://www.dr-chuck.com/page2.htm"

Browser



An HTTP request - response cycle

Making an HTTP request

- Connect to the server
 - a "hand shake"
- Request a document (or the default document)
 - GET `http://dr-chuck.com/page1.htm`
 - GET `http://www.mlive.com/ann-arbor/`
 - GET `http://www.facebook.com`

```
$ telnet www.dr-chuck.com 80
Trying 74.208.28.177...
Connected to www.dr-chuck.com.
Escape character is '^J'.
GET http://www.dr-chuck.com/page1.htm
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>.
</p>
Connection closed by foreign host.
```

Getting Data From The Server

- Each the user clicks on an anchor tag with an href= value to switch to a new page, the browser makes a connection to the web server and issues a "GET" request - to GET the content of the page at the specified URL
- The server returns the HTML document to the Browser which formats and displays the document to the user.

Network Working Group
Request for Comments: 1945
Category: Informational

T. Berners-Lee
MIT/LCS
R. Fielding
UC Irvine
H. Frystyk
MIT/LCS
May 1996

Hypertext Transfer Protocol -- HTTP/1.0

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

IESG Note:

The IESG has concerns about this protocol, and expects this document to be replaced relatively soon by a standards track document.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed

5. Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use. For backwards compatibility with the more limited HTTP/0.9 protocol, there are two valid formats for an HTTP request:

```
Request      = Simple-Request | Full-Request
Simple-Request = "GET" SP Request-URI CRLF
Full-Request  = Request-Line           ; Section 5.1
                *( General-Header      ; Section 4.3
                  | Request-Header     ; Section 5.2
                  | Entity-Header )    ; Section 7.1
                CRLF
                [ Entity-Body ]       ; Section 7.2
```

If an HTTP/1.0 server receives a Simple-Request, it must respond with an HTTP/0.9 Simple-Response. An HTTP/1.0 client capable of receiving a Full-Response should never generate a Simple-Request.

5.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The

```
$ telnet www.facebook.com 80
```

Trying 69.63.187.19...

Connected to www.facebook.com.

Escape character is '^['.

```
GET /
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
```

```
id="facebook" class="no_js">
```

```
<head>
```

```
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
```

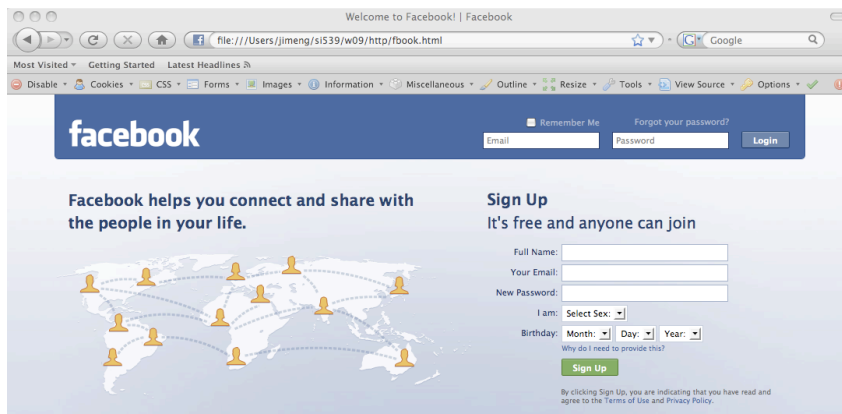
```
<meta http-equiv="Content-language" content="en" />
```

```
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
```

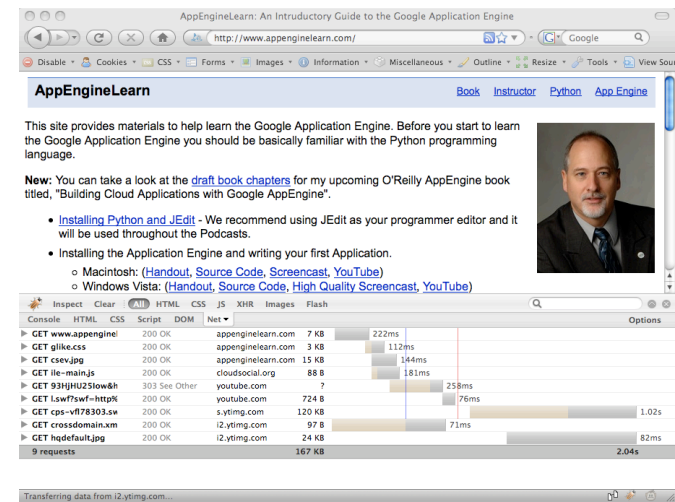
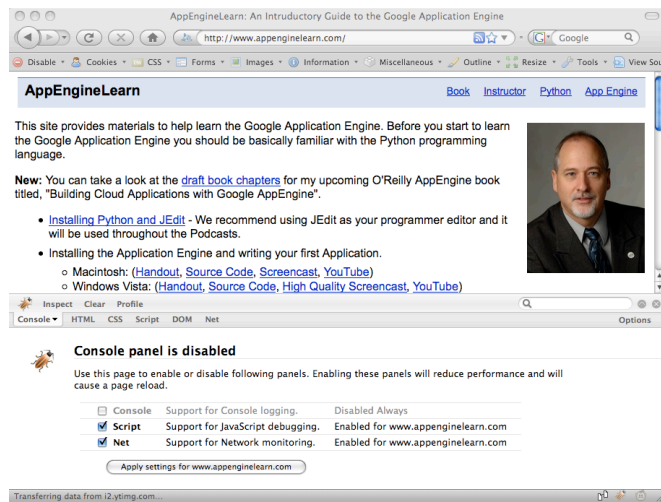
```
<script type="text/javascript">
```

Firebug helps again

- If you haven't already installed Firebug, you need it now
- It can help explore the HTTP request-response cycle
- Some simple-looking pages involve lots of requests:
 - HTML page(s)
 - Image files
 - CSS Style Sheets
 - Javascript files

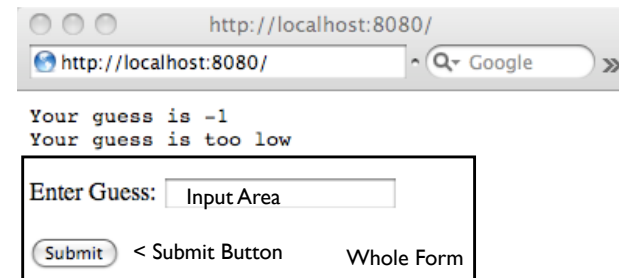


Hmmm - This looks kind of Complex.. Lots of GET commands



Sending Data to an Application

Forms - Input on the Web



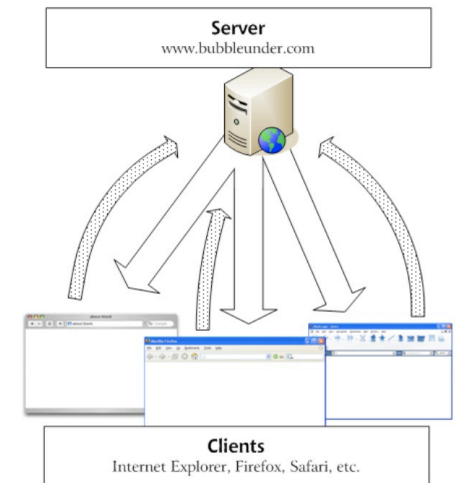
Why do we call them forms?

Forms Need Servers

- Forms effectively gather data from the user and “submit” it to a web page on a server
- The earliest form of server-side processing was called GGI - Which stood for Common Gateway Interface
- CGI allows software to “receive” the input parameters and produce the HTML response - rather than simply reading the HTML content from a file

http://en.wikipedia.org/wiki/Common_Gateway_Interface

- Typical Server
 - Permanently connected to the network
 - Has static address
 - Is part of a cloud
- Clients
 - Browsers
 - Many clients at the same time using the server

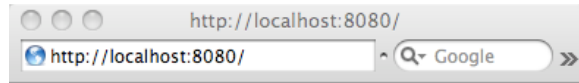


Using Forms Without Servers

- Submitting form data works without a server - the browser moves from static page to static page
- But the data from the forms is neither saved nor is it usable

Pressing Submit (Get)

- When you fill in a form and press “Submit” the browser packs up the parameters of the form and sends them to the server using the “name=” values as the parameter names and the field contents as the values.
- Then this request returns new HTML which is shown in the browser.



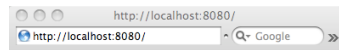
Your guess is -1
Your guess is too low

Enter Guess: 20

Submit

SERVER

So lets write some code and make
a server...



Your guess is -1
Your guess is too low

Enter Guess: 20

Submit

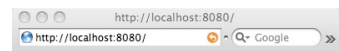
print "Your guess is", guess

answer = 42

if guess < answer :
 print "Your guess is too low"

if guess == answer :
 print "Congratulations!"

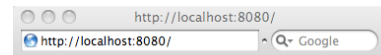
if guess > answer :
 print "Your guess is too high"



Your guess is 20
Your guess is too low

Enter Guess: 25

Submit



Your guess is -1
Your guess is too low

Enter Guess:

Submit

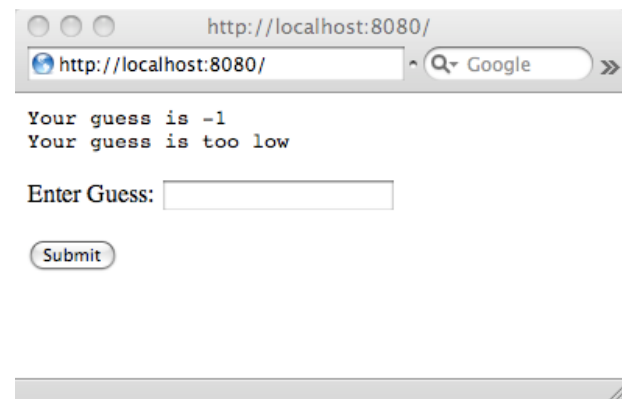
```
<p>Your guess is 20.</p>
<p>Your guess is too low.</p>
<form method="post" action="/">
  <p>
    Enter Guess:
    <input type="text" name="guess" />
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
```



```
<p>Your guess is 20.</p>
<p>Your guess is too low.</p>
<form method="post" action="/">
  <p>
    Enter Guess:
    <input type="text" name="guess" />
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
```

```
<p>Your guess is 20.</p>
<p>Your guess is too low.</p>
<form method="post" action="/">
  <p>
    Enter Guess:
    <input type="text" name="guess" />
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
```

```
<p>Your guess is 20.</p>
<p>Your guess is too low.</p>
<form method="post" action="/">
  <p>
    Enter Guess:
    <input type="text" name="guess" />
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
```



Attributes of a form element

- "action" attribute tells where to submit the form
 - Usually the path to a script or program on the server that processes the form inputs
- "method" attribute tells how to submit the form
 - In this case using HTTP POST
 - See page 30 of RFC 1945

```
<p>Your guess is 20.</p>
<p>Your guess is too low.</p>
<form method="post" action="/">
  <p>
    Enter Guess:
    <input type="text" name="guess" />
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
```

GET .vs. POST

- Two ways the browser can send parameters to the web server
 - GET - Parameters are placed on the URL which is retrieved
 - POST - The URL is retrieved and parameters are appended to the request in the the HTTP connection

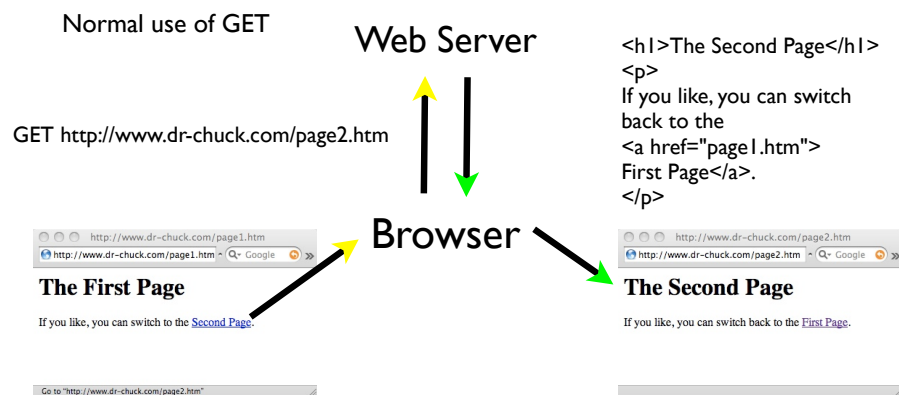
<nerdy-stuff>

Intended purpose of POST

- Posting a message to a bulletin board, newsgroup, mailing list
- Annotation of existing resources
- Extending a database through an append operation
- Creating a new object
- Providing a block of data, such as the result of submitting a form

As opposed to GET

- Retrieve a resource identified by the path portion of the URL



Browser `<input type="text" name="guess" />`

Web Server



Browser

`<input type="text" name="guess" />`

Web Server



HTTP
Request

Browser

POST /
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 8
guess=25

`<input type="text" name="guess" />`

Passing Parameters with GET

Web Server



HTTP
Request

Browser

GET /simpleform.html?guess=25
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14

POST /simpleform.html
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 13
guess=25

`<input type="text" name="guess" />`

`</nerdy-stuff>`

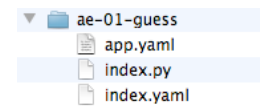
“Rules” for GET and POST

- GET is used when you are reading or searching things
- POST is used when data is being created or modified
- Web search spiders will follow GET URLs but generally not POST URLs
- GET URLs should be “idempotent” - the same URL should give the “same thing” each time you access it
- GET has an upper limit of the number of bytes of parameters and values (think about 2K)

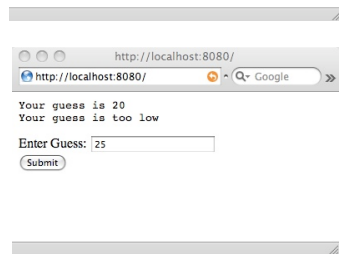
Writing Your AppEngine Application

Application Folder


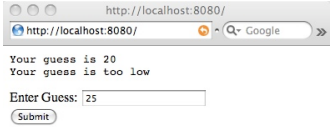
- app.yaml - Defines the name of your application and the high level routing of incoming URLs
- index.py - The code for your application
- index.yaml - Created by App Engine



```
print "Your guess is", guess  
answer = 42  
  
if guess < answer :  
    print "Your guess is too low"
```



```
if guess == answer :  
    print "Congratulations!"  
  
if guess > answer :  
    print "Your guess is too high"
```

index.py

```

print "Your guess is", guess
answer = 42

if guess < answer :
    print "Your guess is too low"

if guess == answer :
    print "Congratulations!"

if guess > answer :
    print "Your guess is too high"

```

app.yaml

app.yaml

- The app.yaml file routes requests amongst different Python scripts.

```

application: ae-01-guess
version: 1
runtime: python
api_version: 1

handlers:
- url: /*
  script: index.py

```

index.py

```

import sys

print 'Content-Type: text/html'
print ""
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
    guess = int(data[data.find('=')+1:])
except:
    guess = -1
    print 'Your guess is too high'

```

Web Server

HTTP
Request

↑

Browser

```

POST /
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 8
guess=25

```

<input type="text" name="guess" />

```
import sys
```

```
print 'Content-Type: text/html'
print "
print '<pre>'
```

```
# Read the form input which is a single line as follows
```

```
# guess=42
```

```
data = sys.stdin.read()
```

```
# print data
```

```
try:
```

```
    guess = int(data[data.find('=')+1:])
```

```
except:
```

```
    guess = -1
```

```
    print 'Your guess is too high'
```

```
import sys
```

```
print 'Content-Type: POST /
print "Accept: www/source
```

```
print "Accept: text/html
```

```
print '<pre>'User-Agent: Lynx/2.4 libwww/2.14
```

```
Content-type: application/x-www-form-urlencoded
```

```
# Read the form input
Content-length: 8
```

```
# guess=42
guess=25
```

```
data = sys.stdin.read()
```

```
# print data
```

```
try:
```

```
    guess = int(data[data.find('=')+1:])
```

guess=25

```
except:
```

```
    guess = -1
```

```
    print 'Your guess is too high'
```

guess=25

```
guess = int(data[data.find('=')+1:])
```

guess=25

5

```
guess = int(data[data.find('=')+1:])
```

guess=25

5 6

```
guess = int(data[data.find('=')+1:])
```

guess=

5 6

```
guess = int(data[data.find('=')+1:])
```

```
import sys
```

```
print 'Content-Type: text/html'
```

```
print "
```

```
print '<pre>'
```

```
# Read the form input which is a single line as follows
```

```
# guess=42
```

```
data = sys.stdin.read()
```

```
# print data
```

```
try:
```

```
    guess = int(data[data.find('=')+1:])
```

guess=25

```
except:
```

```
    guess = -1
```

```
    print 'Your guess is too high'
```

```
print 'Your guess is', guess
```

```
answer = 42
```

```
if guess < answer :
```

```
    print 'Your guess is too low'
```

```
if guess == answer:
```

```
    print 'Congratulations!'
```

```
if guess > answer :
```

```
    print 'Your guess is too high'
```

```
print '</pre>'
```

```
print "<form method='post' action='/'>
```

```
<p>Enter Guess: <input type="text" name="guess"/></p>
```

```
<p><input type="submit"></p>
```

```
</form>"
```



```
print 'Your guess is', guess
```

```
answer = 42
```

```
if guess < answer :
```

```
    print 'Your guess is too low'
```

```
if guess == answer:
```

```
    print 'Congratulations!'
```

```
if guess > answer :
```

```
    print 'Your guess is too high'
```

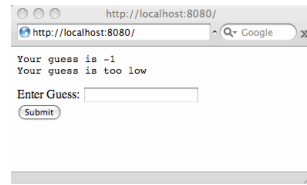
```
print '</pre>'
```

```
print "<form method='post' action='/'>
```

```
<p>Enter Guess: <input type="text" name="guess"/></p>
```

```
<p><input type="submit"></p>
```

```
</form>"
```



Advanced Form Fields

Fieldset and Legend

```
<form method="get" action="simpleform.html">
```

```
<fieldset>
```

```
<legend>All About You</legend>
```

```
<p>
```

```
<label for="yourname">Enter your name:</label>
```

```
<input type="text" name="yourname" id="yourname" />
```

```
</p>
```

```
<p><input type="submit" /></p>
```

```
</fieldset>
```

```
</form>
```



Input Types

- Text
- Password
- Checkbox
- Radio Button
- Hidden
- Submit
- File

Text Input

```
<p>
<label for="nameinp">Enter your name:</label>
<input type="text" name="yourname" id="nameinp" />
</p>
<p>
<label for="nickinp">Enter your nickname:</label>
<input type="text" name="nickname" id="nickinp" value="Bob" />
</p>
```

The id= attribute is used to reference the field inside the HTML document. The name= attribute is the parameter name used to submit the data to the server.

Text fields can either start out blank or have content pre-populated.

Password Input Type

```
<p>
<label for="password">Your password:</label>
<input type="password" id="password" name="password" />
</p>
```

Your password:

This only hides the password from view on the screen - to protect the password while in-transit, you need to send the data over https.

Hidden

- Hidden fields are used generally so that a program in a web server can send some internal information back to itself.

```
<input type="hidden" name="peekaboo" value="hereiam" />
```

Checkbox - Multiple Select

```
<p>
<input type="checkbox" name="terms" id="termid" />
<label for="termid">I have read the terms and conditions.</label>
</p>
<p>
<input type="checkbox" name="offers" id="offerid" />
<label for="offerid">I agree that you can contact me regarding
special offers in the future.</label>
</p>
```

☒ I have read the terms and conditions.

☐ I agree that you can contact me regarding special offers in the future.

Checkbox - Preselected

```
<p>
<input type="checkbox" name="terms" id="termid" />
<label for="termid">I have read the terms and conditions</label>
</p>
<p>
<input type="checkbox" name="offers" id="offerid" checked="checked" />
<label for="offerid">I agree that you can contact me regarding
special offers in the future</label>
</p>
```

Radio Buttons - Choice

- ☒ In the morning
☐ In the afternoon
☐ In the evening

```
<p>
<input type="radio" name="timeslot" id="mor" value="morning" checked="checked" />
<label for="mor">In the morning</label>
<br />
<input type="radio" name="timeslot" id="aft" value="afternoon" />
<label for="aft">In the afternoon</label>
<br />
<input type="radio" name="timeslot" id="eve" value="evening" />
<label for="eve" ">In the evening</label>
</p>
```

Drop Down List

Which best describes you?

```
<p>
<label for="role">Which best describes you?</label>
<select name="role" id="role">
<option value="1">Secretary</option>
<option value="2" selected="selected">Web Designer</option>
<option value="3">Manager</option>
<option value="4">Cleaner</option>
<option value="5">Other</option>
</select>
</p>
```

A drop-down list generates a single value when it is sent to the server.

role=2

Textarea for paragraphs

Please tell us about your hobbies:

```
<p>
<label for="hobbies">Please tell us about your hobbies:</label>
</p>
<p>
<textarea name="hobbies" rows="7" cols="40" id="hobbies">
Old Value
</textarea>
</p>
```

Textareas can become rich text areas - <http://tinymce.moxiecode.com/>

File Uploads

```
<input type="file" name="datain" accept="text/html">
```

- File input is simple on the browser
- You can optionally insist on only certain file types
- File input processing depends on which software is receiving the file input on the server

Submit Button(s)

When you have multiple submit buttons the value can be used to figure out which button was pressed.

```
<p><input type="submit"/></p>
```

```
<!-- Multiple submit buttons -->
```

```
<p>
```

```
<input type="submit" name="subtype" value="Submit"/>
```

```
<input type="submit" name="subtype" value="Cancel"/>
```

```
</p>
```

Parameter

http:// ... /url?subtype=Submit

http:// ... /url?subtype=Cancel

Value

Dumper Program: Working in the Depths of the Machine

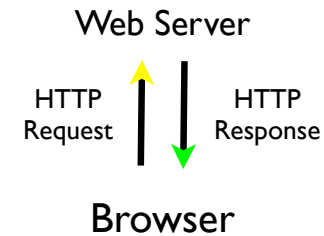
Dumper

- This application makes a form and then we submit the form via POST
- This application dumps the input variables that come in from the HTTP Request
- This is the most basic view of a web application
- It is like a Dinosaur - it is a web program written using the "old ways"

Common Gateway Interface

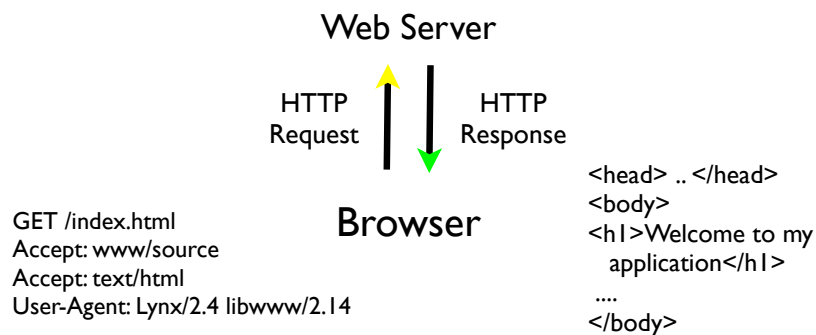
- A set of rules about taking input parameters and data from an incoming HTTP request and handing them to the program.

HTTP Request / Response Cycle



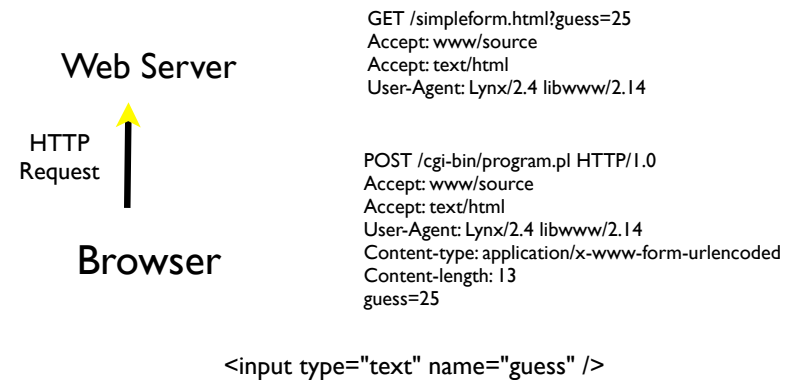
http://www.oreilly.com/openbook/cgi/ch04_02.html

HTTP Request / Response Cycle



http://www.oreilly.com/openbook/cgi/ch04_02.html

Passing Parameters to The Server





The Common Gateway Interface

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

The current version is CGI/1.1.

CGI Documentation

If you have no idea what CGI is, you should read this [introduction](#).

Once you have a basic idea of what CGI is and what you can use it for, you should read this [primer](#) which will help you get started writing your own gateways.

If you are interested in handling the output of [HTML forms](#) with your CGI program, you will want to read this guide to [handling forms with CGI programs](#).

Security is a crucial issue when writing CGI programs. Please read these [tips](#) on how to write CGI programs which do not allow malicious users to abuse them.

When you get more advanced, you should read the [interface specification](#) which will help you utilize CGI to the fullest extent. If you are a server software author, it will help you add CGI compliance to your information server.

<http://hoohoo.ncsa.uiuc.edu/cgi/in.html>

CGI In App Engine

- When a Request is received in the App Engine, according to the rules of CGI
- The environment variables such as server name, document path, etc come in a dictionary object
- Any POST data comes in on standard input (sys.stdin)
- Whatever the program prints goes to the browser as the HTTP response

B
r
o
w
s
e
r

```
<form method="post" action="/">
```

```
Zap Data: <input type="text" name="zap"><br>
```

```
Zot Data: <input type="text" name="zot"><br>
```

```
<input type="submit">
```

```
</form>
```

Zap Data:

Zot Data:

Environment
Data

CGI
Program

dict()

POST Data

zap=Stuff&zot=MoreStuff

stdin

```
<form method= ....>
</form>
```

print

```
import os
import sys
```

The standard output (print
statements) produce the HTTP
Response.

```
print 'Content-Type: text/html'
print ''
print '<form method="post" action="/">'
print 'Zap Data: <input type="text" name="zap"><br>'
print 'Zot Data: <input type="zot" name="zot"><br>'
print '<input type="submit">'
print '</form>'
```

The output consists of HTTP
headers followed by the body of
the document.

Zap Data:

Zot Data:

import sys

- <http://docs.python.org/library/sys.html>

```
sys.stdin
sys.stdout
sys.stderr
```

File objects corresponding to the interpreter's standard input, output and error streams. `stdin` is used for all interpreter input except for scripts but including calls to `input()` and `raw_input()`. `stdout` is used for the output of `print` and `expression` statements and for the prompts of `input()` and `raw_input()`. The interpreter's own prompts and (almost all of) its error messages go to `stderr`. `stdout` and `stderr` needn't be built-in file objects; any object is acceptable as long as it has a `write()` method that takes a string argument. (Changing these objects doesn't

import os

- <http://docs.python.org/index.html> -- Library Reference
- <http://docs.python.org/library/os.html>

Process Parameters

These functions and data items provide information and operate on the current process and user.

os.environ

A mapping object representing the string environment. For example, `environ['HOME']` is the pathname of your home directory (on some platforms), and is equivalent to `getenv("HOME")` in C.

This mapping is captured the first time the `os` module is imported, typically during Python startup as part of processing `site.py`. Changes to the environment made after this time are not reflected in `os.environ`, except for changes made by modifying `os.environ` directly.

If the platform supports the `__environ__` function, this mapping may be used to modify the environment as well as

```
print 'Environment keys:'
print ''
for param in os.environ.keys():
    print param, ":", os.environ[param]
print ''
```

```
Environment keys:
HTTP_COOKIE : camtoolspref=
SERVER_SOFTWARE : Development/1.0
SCRIPT_NAME :
REQUEST_METHOD : GET
PATH_INFO : /
SERVER_PROTOCOL : HTTP/1.0
QUERY_STRING :
CONTENT_LENGTH :
HTTP_USER_AGENT : Mozilla/5.0 (Macintosh; U; Inte
HTTP_CONNECTION : keep-alive
SERVER_NAME : localhost
REMOTE_ADDR : 127.0.0.1
PATH_TRANSLATED : /Users/csev/Desktop/teach/a539-
SERVER_PORT : 8081
AUTH_DOMAIN : gmail.com
CURRENT_VERSION_ID : 1.1
HTTP_HOST : localhost:8081
TZ : UTC
HTTP_CACHE_CONTROL : max-age=0
USER_EMAIL :
HTTP_ACCEPT : text/xml,application/xml,application
APPLICATION_ID : ae-02-dumper
GATEWAY_INTERFACE : CGI/1.1
HTTP_ACCEPT_LANGUAGE : en-us
CONTENT_TYPE : application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING : gzip, deflate
```

```
print 'Data'
count = 0
for line in sys.stdin:
    count = count + 1
    print line
    if count > 100:
        break
print '</pre>'
```

Reading the POST Data
Spaces are encoded as +

```
HTTP_ACCEPT_LANGUAGE : en-us
CONTENT_TYPE : application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING : gzip, deflate
```

```
Data
zap=Stuff&zap=More+Stuff
```

Up Next: webapp Framework

- While we could write our application using the low-level data provided to our Python code, this would become very tedious
- We would constantly be reading a lot of Internet Standards documents

```
Environment keys:
HTTP_COOKIE : camtoolspref=
SERVER_SOFTWARE : Development/1.0
SCRIPT_NAME :
REQUEST_METHOD : GET
PATH_INFO : /
SERVER_PROTOCOL : HTTP/1.0
QUERY_STRING :
CONTENT_LENGTH :
HTTP_USER_AGENT : Mozilla/5.0 (Macintosh; U; Inte
HTTP_CONNECTION : keep-alive
SERVER_NAME : localhost
REMOTE_ADDR : 127.0.0.1
PATH_TRANSLATED : /Users/csev/Desktop/teach/a539-
SERVER_PORT : 8081
AUTH_DOMAIN : gmail.com
CURRENT_VERSION_ID : 1.1
HTTP_HOST : localhost:8081
TZ : UTC
HTTP_CACHE_CONTROL : max-age=0
USER_EMAIL :
HTTP_ACCEPT : text/xml,application/xml,application
APPLICATION_ID : ae-02-dumper
GATEWAY_INTERFACE : CGI/1.1
HTTP_ACCEPT_LANGUAGE : en-us
CONTENT_TYPE : application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING : gzip, deflate
```

Summary

- We can present our user a form with fields to fill in.
- The data from the form can be sent to the server of our choice
- We write an application on the server which received the data and produces a response